Local Covering: Adaptive Rule Generation Method Using Existing Rules for XCS

Masakazu Tadokoro*, Satoshi Hasegawa*, Takato Tatsumi*, Hiroyuki Sato*, and Keiki Takadama*
*The University of Electro-Communications
Tokyo, Japan
Email: {tadokoro@, hasegawas@cas.lab., tatsumi@, sato@hc., keiki@inf.}uec.ac.jp

***Abstract*—This paper focuses on the covering mechanism in Learning Classifier System (LCS) which generates a new classifier (i.e., an if-then rule) when no classifier matches the input. We propose Local Covering, a niche-based rule generation method for XCS, to reduce the sensitivity of a hyperparameter that determines the generalization probability of initial rules. In Local Covering, the system looks up classifiers in the population and finds the closest match from the input. After that, the selected classifier is copied as the covering classifier and its condition is generalized to cover the input. To integrate the Local Covering method with XCS, XCS-LCPCI (XCS with Local Covering for Previously Covered Inputs) is proposed, which executes Local Covering only if the input has been previously covered by the covering processes. The experimental results with three different problems show that XCS-LCPCI successfully acquires the general classifiers with any of six different parameter settings, while the performance of the conventional XCS varies depending on the parameter settings.**

***Index Terms*—learning classifier system, XCS, covering, MNIST problem**

I. Introduction

Learning Classifier System (LCS) [1] is a paradigm of machine learning, which integrates Genetic Algorithm (GA) and reinforcement learning to generate if-then rules called *classifiers* that match multiple inputs. LCS aims to acquire a maximally-generalized mapping of the states (inputs) with the actions (outputs) according to the rewards (i.e., correct/incorrect). Among LCSs, XCS (eXtended Classifier System) [2] is the most studied variant whose classifier fitness is based on the accuracy of payoff prediction. In general, XCS deals with binary data, and the condition of a classifier consists of the ternary symbols {0,1,#}, where "#" (don't care symbol) matches both 0 and 1 in the corresponding attribute. Such interpretability of XCS is widely shown in many data mining problems. For example, Kharbat *et al.* [3] reported that XCS achieved higher mining performance than C4.5 in a medical data mining task.

However, XCS has several sensitive parameters to be tuned by hand. In particular, the parameter $P_{\#}$ strongly influences the performance of XCS. $P_{\#}$ determines the probability of inserting "#" to the classifier in the initial rule generation operation called *covering*, and its most suitable value varies depending on the problem. If $P_{\#}$ is set far from the "#" rate of desired optimal rules, the learning performance of XCS decreases. To tackle this problem, Fredivianus *et al.* [4] has proposed XCS-RC, which employs a heuristic-based rule combining method as a genetic operator instead of GA. XCS-RC generates no "#" in the covering operation, and it does not need the parameter $P_{\#}$ to be tuned. However, XCS-RC has three additional hyperparameters for rule combining and these hyperparameters need to be tuned considering the problem.

Towards no hyperparameters in the covering operation in a *true sense*, this paper proposes a novel covering method Local Covering, which references existing classifiers in the population [P] to determine where the "#" symbol should be generated in the condition of the covering classifier instead of a fixed probability parameter $P_{\#}$ in XCS. Since Local Covering requires the classifiers in the population [P], it is integrated with XCS by executing Local Covering only if the input of the classifier has ever been previously covered; otherwise, the conventional covering method with $P_{\#}$ is operated. Here, this paper calls the above improved XCS, XCS with Local Covering for Previously Covered Inputs (XCS-LCPCI). To investigate the effectiveness of XCS-LCPCI, this paper conducts the experiments of the three different problems with the several different $P_{\#}$ settings, and shows how XCS-LCPCI is robust to the $P_{\#}$ settings from these results.

The rest of this paper is organized as follows: Section II introduces an overview of XCS; Section III explains our proposed method Local Covering and how it is integrated with XCS as XCS-LCPCI; Section IV conducts the three experiments to evaluate our method; Section V shows the experimental results; Section VI discusses the implications from the results; and finally, Section VII concludes the paper along with future work.

II. XCS Classifier System

A. Overview

XCS is a machine learning system that evolves if-then rules named *classifiers* by GA through reinforcement learning. A classifier mainly consists of the condition C, the action A, the prediction p, the prediction error ϵ, the fitness F. XCS updates C and A by the mutation/crossover operators of GA. The prediction p is updated to be closer to the received reward, and the prediction error ϵ indicates the difference between p and the actual reward.

Fig. 1 shows the architecture of XCS. [P] is the population, which stores all classifiers in it. When XCS receives an input state from the environment, the system forms the match set [M] of classifiers whose condition matches the state.

If [M] does not have θ_{mna} or more different actions, the system generates new classifiers called *covering classifiers* that matches the input state and has an action not appeared in [M]. This operation is called *covering*, and [M] has at least θ_{mna} different actions by this. The system forms prediction array PA, where the payoff prediction of each action is recorded, by calculating the weighted average of p with the weight of F. On the basis of the payoff prediction in PA, the output action is chosen by epsilon-greedy selection. After the action selection, XCS receives the reward from the environment as an evaluation value of the action execution. To update classifiers that contributed to the action selection, the system forms the action set [A] by picking up classifiers that have the same action as the selected one from [M]. After [A] is formed, the prediction p, prediction error ϵ, and fitness F of classifiers in [A] are updated by referring to the received reward.

B. Covering

The covering is a classifier generation operation invoked when [M] does not have θ_{mna} different actions. Each attribute of the input state is changed to "#" with a fixed probability $P_{\#}$. Since $P_{\#}$ determines how many "#"s are generated in the initial iterations, it needs to be tuned considering the problem. Butz *et al.* [5] suggests setting $P_{\#}$ to a value around 0.33, however, the optimal setting is totally depending on the problem.

III. Local Covering

Local Covering is a covering method that generates a new condition based on a classifier selected from the population [P]. In Local Covering, the symbols in the condition different from the input are replaced with "#" to generate a new condition. This method aims to suppress the generation of overgeneral classifiers in the covering operation. These overgeneral classifiers could result in less covering occurred in later iterations, and that interferes with learning convergence especially when rules should have a small number of "#". To avoid the generation of overgeneral classifiers, Local Covering generates the minimum number of additional "#" to cover the unseen input.

A. Procedure of Condition Generation

In Local Covering, a new classifier is generated by the following procedure:

1) The *Minimum Hamming Distance* (MHD, described later) between the input state σ and the condition C is calculated for all classifiers in [P].
2) The classifiers having the minimum value of MHD are chosen from the population [P] under several conditions. The condition C of chosen classifiers are stored in an array $selectedConds$.
3) The condition of a new classifier is randomly chosen from $selectedConds$.
4) The symbols other than "#" are replaced with "#" if the corresponding attribute is different from the input σ.

Note that the conventional covering method that is operated using $P_{\#}$ if the array $selectedConds$ is empty.

Fig. 2 shows how Local Covering generates a covering classifier using existing classifiers in [P]. In this example, the system tries to generate a covering classifier with the action "0". The condition "1110#1" is selected as the nearest one from the input state "111101", and the fourth attribute of "1110#1" is replaced with "#" so that the condition of the covering classifier covers the state. As a result, the condition of the covering classifier is set to "111##1". Since Local Covering iterates over [P] to select the nearest condition and MHD iterates over attributes of the condition, the time complexity of this algorithm is O(mn), where m is the data length and n is the number of classifiers in [P].

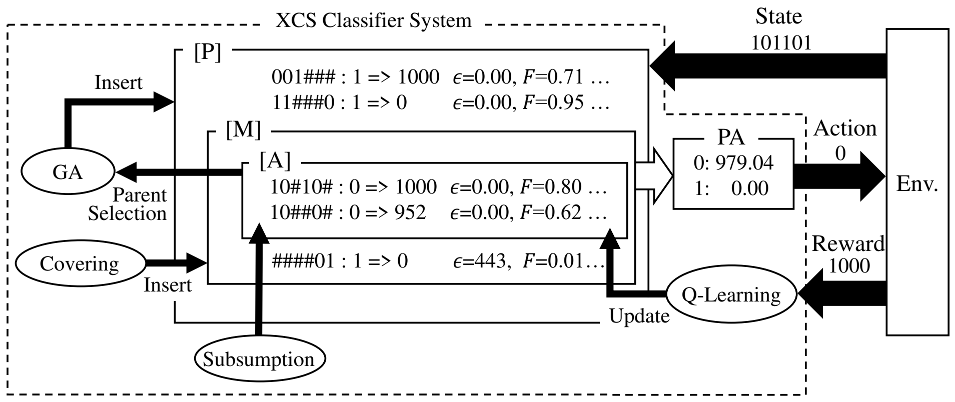
To apply Local Covering, the function GenerateCovering-Classifier in [5] is replaced with Algorithm 1. Note that StandardCovering at line 15 is the conventional covering function of XCS.

Algorithm 1 LocalCovering([P], [M], σ)

```
1:  $selectedConds \leftarrow$ empty array
2:  $D \leftarrow$ +inf
3:  $a \leftarrow$ random action not present in [M]
4:  **for each** classifier $cl$ in [P] **do**
5:      $d \leftarrow$ MinimumHammingDistance($cl.C, \sigma$)
6:      **if** $cl.A = a$ and $cl.exp \geq 1$ and $d \leq D$ **then**
7:          **if** $d < D$ **then**
8:              $D \leftarrow d$
9:              $selectedConds \leftarrow$ empty array
10:         **end if**
11:         $selectedConds$.append($cl.C$)
12:     **end if**
13: **end for**
14: **if** $selectedConds$ is empty **then**
15:     **return** StandardCovering([M], $\sigma$)
16: **end if**
17: $Cl \leftarrow$ new classifier
18: $Cl.A \leftarrow a$; $Cl.p \leftarrow p_I$; $Cl.\epsilon \leftarrow \epsilon_I$; $Cl.F \leftarrow F_I$;
    $Cl.exp \leftarrow 0$; $Cl.as \leftarrow 1$; $Cl.num \leftarrow 1$;
    $Cl.ts \leftarrow$ current iteration count;
    $Cl.C \leftarrow$ random in $selectedConds$
19: **for** $i = 0...(\#\sigma$ - 1) **do**
20:     **if** $Cl.C[i] \neq$ "#" and $Cl.C[i] \neq \sigma[i]$ **then**
21:         $Cl.C[i] \leftarrow$ "#"
22:     **end if**
23: **end for**
24: **return** $Cl$
```

B. Minimum Hamming Distance (MHD)

To determine how close a condition C is to the input state σ, we introduce *Minimum Hamming Distance* (MHD), which uses the hamming distance between the input state and the closest state covered in the condition. In the ternary representation, MHD denotes how many attributes in the condition C are required to be replaced with "#" to cover



XCS Classifier System
State
101101
[P]
001### : 1 => 1000 ϵ=0.00, F=0.71 …
11###0 : 1 => 0 ϵ=0.00, F=0.95 …
Insert
[M]
[A]
10#10# : 0 => 1000 ϵ=0.00, F=0.80 …
10##0# : 0 => 952 ϵ=0.00, F=0.62 …
####01 : 1 => 0 ϵ=443, F=0.01…
GA
Parent Selection
Covering
Insert
Update
Subsumption
PA
0: 979.04
1: 0.00
Action
0
Env.
Reward
1000
Q-Learning

Fig. 1: The Architecture of the XCS Classifier System

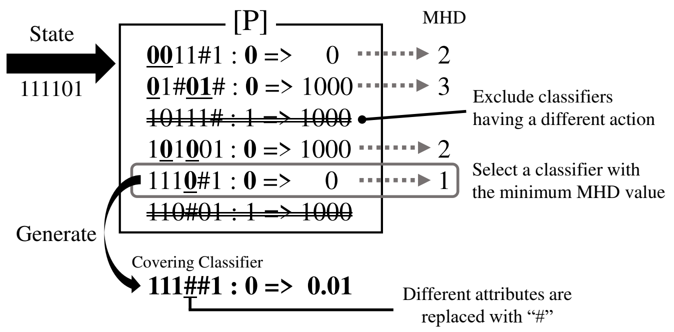


Fig. 2: Classifier Generation Procedure of Local Covering

the input state σ. MHD is calculated by Algorithm 2, where C denotes a condition and σ denotes an input state.

Algorithm 2 MinimumHammingDistance(C, σ)

```
1: $d \leftarrow 0$
2: for $i = 0...(\#\sigma - 1)$ do
3:     if $C[i] \neq$ "#" and $C[i] \neq \sigma[i]$ then
4:         $d \leftarrow d + 1$
5:     end if
6: end for
7: return $d$
```

C. XCS with Local Covering for Previously Covered Inputs

XCS with Local Covering for Previously Covered Inputs (XCS-LCPCI) is an XCS that employs Local Covering only in the second or later covering of a state. Since Local Covering refers to existing classifiers and chooses one having the nearest condition, the population [P] should have enough number of classifiers before Local Covering is performed. If Local Covering is applied to the entire covering operation, the input state is compared to dissimilar conditions at an early stage, which results in overgeneralization of the condition. To avoid this situation, XCS-LCPCI introduces a covering classifier set [CC] that stores previously-generated covering classifiers to determine whether to apply Local Covering. In the covering operation of XCS-LCPCI, Local Covering is used if [CC] has a classifier that matches the input state; otherwise, the covering classifier is generated by Standard Covering using $P_{\#}$. Finally, the generated covering classifier is inserted to [CC].

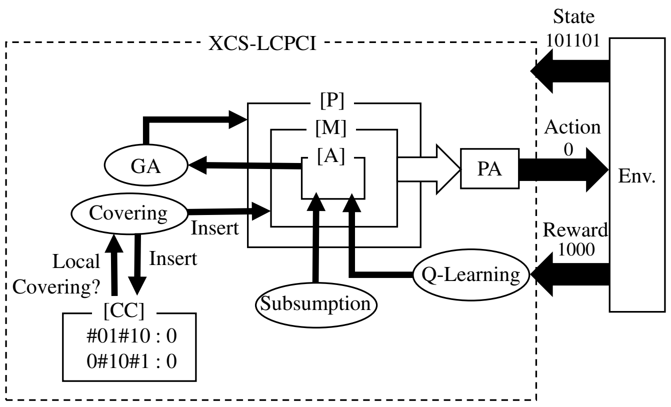


Fig. 3: XCS-LCPCI

Fig. 3 is a schematic diagram of XCS-LCPCI. In this figure, [CC] is the additional component to the original XCS. The upward arrow means XCS-LCPCI determines whether to operate Local Covering by referring to [CC], and the downward arrow means every generated covering classifier is inserted into [CC].

Note that XCS-LCPCI exhibits the same performance as XCS until a previously-generated covering classifier is deleted. Since Standard Covering generates a condition randomly, generated covering classifiers tend to have an inaccurate condition leading to the deletion of these classifiers. In XCS-LCPCI, Local Covering is called after the system starts to delete these inaccurate classifiers. Since XCS performs classifier deletions only after the number of classifiers exceeds N, the population is guaranteed to have N classifiers when Local Covering is performed.

To use XCS-LCPCI, the function GenerateCoveringClassifier is replaced with Algorithm 3.

Algorithm 3 GenerateCoveringClassifier([P], [M], [CC], σ)

```
1: $prevCovered$ = false
2: for each classifier $cl$ in [CC] do
3:     if DoesMatch($cl$, $\sigma$) then
4:         $prevCovered$ = true
5:         break
6:     end if
7: end for
8: if $prevCovered$ = true then
9:     $Cl$ = LocalCovering([P], [M], $\sigma$)
10: else
11:     $Cl$ = StandardCovering([M], $\sigma$)
12: end if
13: [CC].append($Cl$)
14: return $Cl$
```

IV. Experiment

To compare the performance of XCS and XCS-LCPCI, we conducted experiments using three different problems, 20-bit multiplexer (MUX) problem, 11-bit class-imbalanced multiplexer (CIMUX) problem [6], and MNIST problem [7]. These problems have different distributional features. While the distribution of input is uniform in the 20-bit multiplexer (MUX) problem, the 11-bit class-imbalanced multiplexer (CIMUX) problem has a strong bias in class labels and attributes that determine the answer class. MNIST problem is a problem using a dataset of handwritten digits, and it has a large number of input attributes. In addition, in the MNIST problem, only specific attributes (the center part of an image) have importance for classification because the outer part of an image is always "0" (white). The conventional XCS requires different $P_{\#}$ settings for these three problems to obtain optimal performance. Here we use six different $P_{\#}$ settings, $P_{\#} = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$, to compare the robustness of $P_{\#}$ settings of XCS and XCS-LCPCI.

We employ the theoretical parameter settings of XCS [8] for three hyperparameters $\theta_{sub},\ \beta,\ \epsilon_0$ by the problem. To fulfill the condition to use these theoretical parameter settings, Moyenne Adaptive Modifee (MAM) is disabled when updating p and ϵ.

We use the standard parameter setting described in [5] for other hyperparameters, i.e., $\alpha = 0.1,\ \nu = 5,\ \chi = 0.8,\ \mu = 0.04,\ \theta_{del} = 20,\ \delta = 0.1,\ doGASubsumption = \text{true}$. We use our implementation based on XCSJava 1.0 [9].

A. Experiment 1: 20-bit MUX Problem

The MUX problem is a typical benchmark problem of LCS. An input state of the MUX problem consists of a k-bit binary number called address bits and a 2^k-bit binary number called reference bits, where k is a positive integer and a larger value of k makes the problem more difficult. If a denotes the decimal number of the address bits, the answer of this problem is the $(a+1)$-th binary digit of the reference bits.

In this experiment, we use the theoretical parameter settings based on [8], where $\theta_{sub} = 31,\ \beta = 0.1387,\ \epsilon_0 = 58.52847$. We set $N = 2000$ considering the search space size of the problem. Other parameters are set to the standard settings [5]: $doASSubsumption = \text{true},\ \theta_{GA} = 25$.

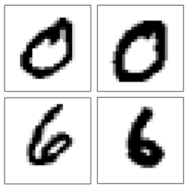


Fig. 4: Examples from the MNIST Dataset

B. Experiment 2: Class-Imbalanced MUX Problem

The Class-Imbalanced MUX (CIMUX) problem [6] is a modified MUX problem that has biases in class labels. In this paper, we choose "0" as a minority class and "1" as a majority class, which means the likelihood of states whose answer is "1" is smaller than that of "0". In the CIMUX problem, the input state is generated randomly, and the state is sent to XCS if the answer is the majority class; otherwise, it is sent to XCS with $1/2^i$ probability, where i denotes the class-imbalance level to determine the difficulty of the problem. The environment repeats to generate a random state until it is sent to XCS in one iteration. The CIMUX problem is difficult in that XCS needs to identify overgeneralized classifiers by a limited number of minority class data.

In this experiment, we set $\theta_{sub} = 15345,\ \beta = 0.0007472,\ \epsilon_0 = 0.130206$ on the basis of the theoretical parameter settings [8]. To reproduce the result of [8], all parameters are set to the same values, i.e., $N = 800,\ doASSubsumption = \text{false},\ \theta_{GA} = 3200$, and the class imbalance level $i = 10$.

C. MNIST Problem

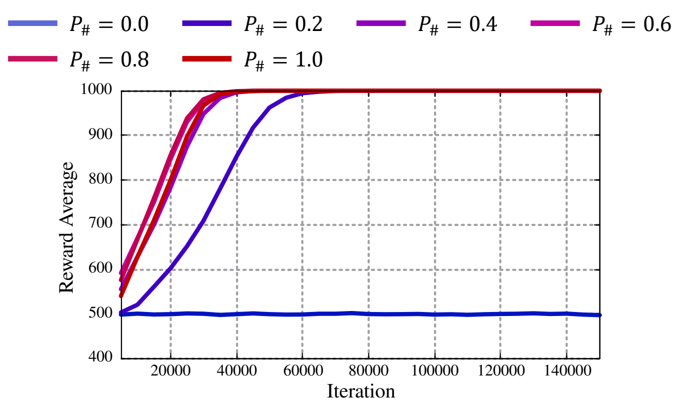
The MNIST problem is a classification task of the handwritten digit images in the MNIST dataset [7]. To make the problem simpler, we use only 11,841 training data and 1,938 test data having a class label of "0" or "6"[1]. To input the image into XCS, each pixel is binarized to "0" (white) or "1" (black), and the image is reshaped to a 784-bit binary string.

To use a theoretical parameter setting based on [8], we assume the correct rate threshold to identify incorrect classifiers is 98% in this problem, and we set $\theta_{sub} = 49,\ \beta = 0.09915,\ \epsilon_0 = 38.34258$. We set $N = 20000$ by referring to the search space size of the problem. Other parameters are set to the standard settings [5]: $doASSubsumption = \text{true},\ \theta_{GA} = 25$.

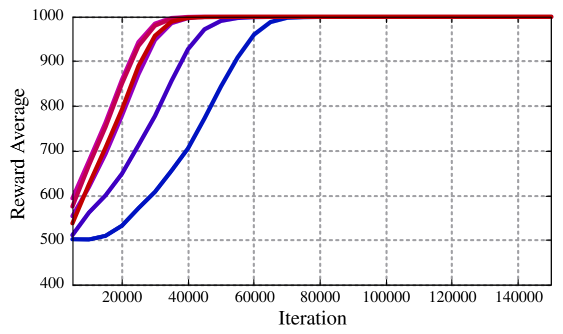
V. Results

Fig. 5, Fig. 6, and Fig. 7 show the plots of the received reward in the 20-bit MUX problem, the 11-bit CIMUX problem, and the MNIST problem, respectively. The results are averaged over 5,000 iterations in Fig. 5 and Fig. 7, and 500,000

[1]XCS can deal with multiclass classification tasks. However, since a large number of classifiers are required to learn the 10-class multiclass classification task of 784-bit inputs appropriately, here we chose this binary classification problem in this experiment.

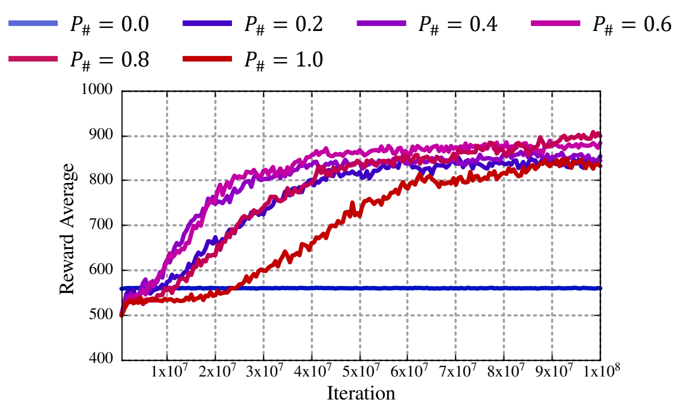


(a) XCS

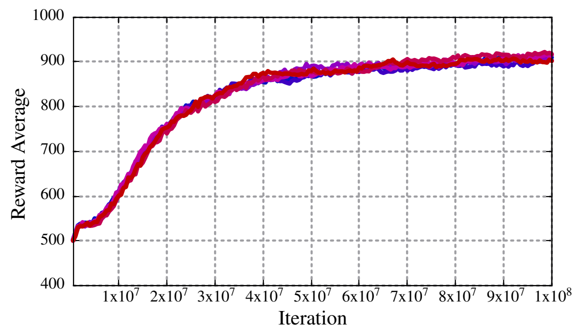


(b) XCS-LCPCI

Fig. 5: Reward Average of Experiment 1 (20-bit MUX)



(a) XCS



(b) XCS-LCPCI

Fig. 6: Reward Average of Experiment 2 (11-bit CIMUX)

iterations in Fig. 6. All these results are averaged over 31 runs using different random seeds.

The results show the performance of XCS is depending on $P_\#$, and especially $P_\# = 0.0, 0.2$ caused a decrease in performance. Especially, XCS can solve the MNIST problem by setting $P_\# = 1.0$ [2], but XCS completely fails to solve it with the other $P_\#$ settings. On the other hand, XCS-LCPCI achieved a comparable performance even with these low $P_\#$ settings. In Fig. 6 (b), XCS-LCPCIs with any $P_\#$ settings show comparable performance to XCS with $P_\# = 0.8$, which achieved the best performance at the last iteration in Fig. 6 (a). This implies XCS-LCPCI has a robustness of both too high $P_\#$ and too low $P_\#$ settings.

VI. Discussion

A. Analysis on Classifiers Generated by Local Covering

Table I shows the rate of "#" for each attribute in covering classifiers generated by Local Covering, where b_i denotes the i-th attribute of the state and a is the decimal number of address bits. b_{4+a} is the reference bit to be an answer to the problem[3]. The result is averaged over 31 different seeds, and the standard deviation is reported in the SD field. This table shows that Local Covering generates less "#" in address bits than in reference bits. In the MUX problems, the desired optimal rules do not have "#" in these address bits. Therefore, this result implies that Local Covering avoids overgeneralization by generating conditions similar to existing ones. On the other hand, the reference bit b_{4+a} does not have a significant difference in the "#" generation rate from other reference bits. From these results, Local Covering can assign "#" to attributes confirmed to be able to generalize by existing classifiers, but it is difficult for Local Covering to find the attribute to specify in the input state because this method replaces all differences between a state and a condition with "#".

Fig. 8 shows examples of covering classifiers generated by the conventional and proposed method in the MNIST problem. In this figure, Standard Covering with $P_\# = 0.0, 0.4$ fails to generate a condition that can be used for other inputs. XCS with $P_\# = 0.0, 0.2, 0.4, 0.6, 0.8$ completely failed to learn the MNIST dataset because [P] is filled with classifiers with too specific conditions generated by Standard Covering. Furthermore, XCS does not update a classifier until the input matches its condition. This causes too specific rules are left until the population size exceeds N, and the system is trapped in a covering/deletion loop. On the other hand, XCS-LCPCI has a mechanism to generalize an existing condition to cover a new input received from the environment, and too specific

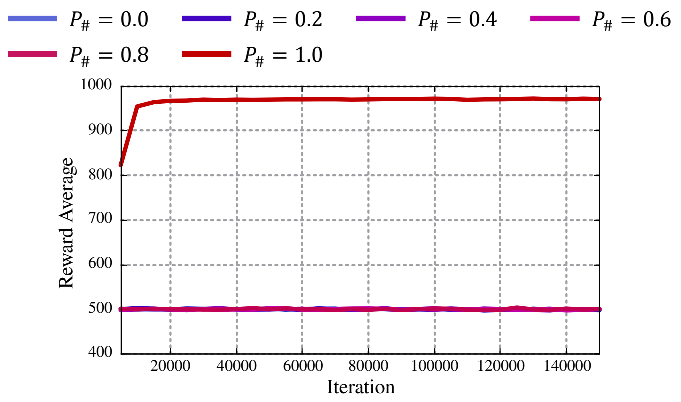
[2]We found the original XCS outperforms the performance of our previous work that uses VAE for preprocessing [10] if $P_\#$ is set to 1.0 in this MNIST problem.

[3]For example, for an input "11101010101", the address bit value is $a = (111)_2 = 7$, and the corresponding reference bit is $b_{4+a} = b_{4+7} = b_{11}$, thus the answer is $b_{11} =$ "1$''$.

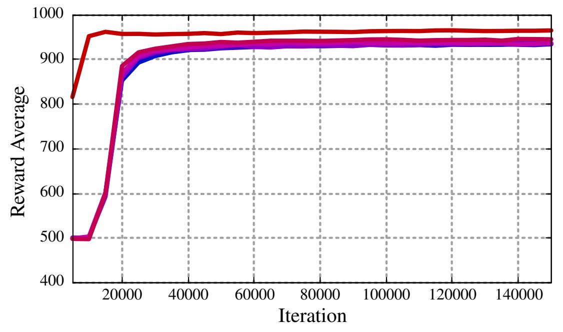
TABLE I: "#" Generation Rates by Local Covering of XCS-LCPCI ($P_\# = 0.8$) in Experiment 2 (11-bit CIMUX)

	b_1	b_2	b_3	b_4	b_5	b_6
"#" generation rate by Local Covering	0.425 (SD: 0.026)	0.419 (SD: 0.025)	0.420 (SD: 0.025)	0.884 (SD: 0.018)	0.885 (SD: 0.018)	0.884 (SD: 0.016)

	b_7	b_8	b_9	b_{10}	b_{11}	b_{4+a}
"#" generation rate by Local Covering	0.884 (SD: 0.019)	0.884 (SD: 0.016)	0.890 (SD: 0.015)	0.886 (SD: 0.020)	0.891 (SD: 0.021)	0.879 (SD: 0.020)



(a) XCS



(b) XCS-LCPCI

Fig. 7: Reward Average of Experiment 3 (MNIST)

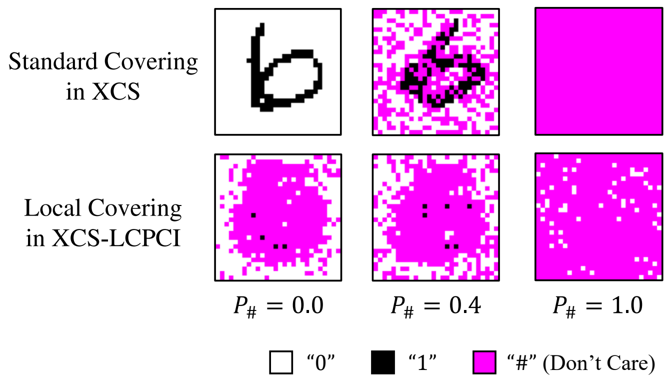


Fig. 8: Examples of Covering Classifiers Generated in Experiment 3

conditions are generalized by using this mechanism.

B. Rule Discovery Strategy of XCS-LCPCI

Generally, the rule exploration strategy of XCS can be statically changed by using different $P_\#$ settings. If $P_\#$ is set to a small value, XCS tries to find a set of rules using a generalization pressure by genetic operators. In contrast, if $P_\#$ is set to a large value such as 1.0, XCS tries to find attributes that can be specified in classifier conditions. However, because of the high-dimensional input of the MNIST problem, a large size of the population is required to find building blocks if the exploration starts with the generalization of specific classifiers using $P_\#$ smaller than 1.0. Local Covering can skip the initial exploration until the first subsumption by forcing the newly-generated classifier condition to include an existing classifier condition even when $P_\#$ is smaller than 1.0. This reduces the generalization cost required for the first subsumption and helps XCS to generalize too specific classifiers in a feasible time. For a better understanding of the difference between Standard Covering and Local Covering, we visualized the transition of classifier conditions in Fig. 9 by calculating weighted average of conditions with the weight of $num \times F \times p$. While the average classifier condition of XCS stays the same, XCS-LCPCI appropriately finds the pixels to generalize in early iterations. This implies that Local Covering skips costly generalization steps by replacing the attributes that vary depending on inputs with "#".

Since XCS-LCPCI switches the covering method to Local Covering as [CC] covers the state-action space, XCS-LCPCI discontinues to use $P_\#$ for covering. This allows the system to escape from a covering/deletion loop caused by the use of a constant probability for the "#" production.

C. Effect of Inheriting the Prediction of the Base Classifier

This section discusses whether to inherit the payoff prediction from the base classifier chosen in Local Covering. Since Local Covering generates a similar classifier to the chosen one from [P], inheriting the prediction value from the base classifier could help covering classifiers to converge their payoff prediction by a smaller number of evaluations.

We investigated an additional experiment to compare XCS-LCPCI with and without the use of prediction inheritance using the 11-bit CIMUX problem with $P_\# = 0.6$. Other settings are set to the same as Experiment 2. In this experiment, the prediction error ϵ is not inherited.

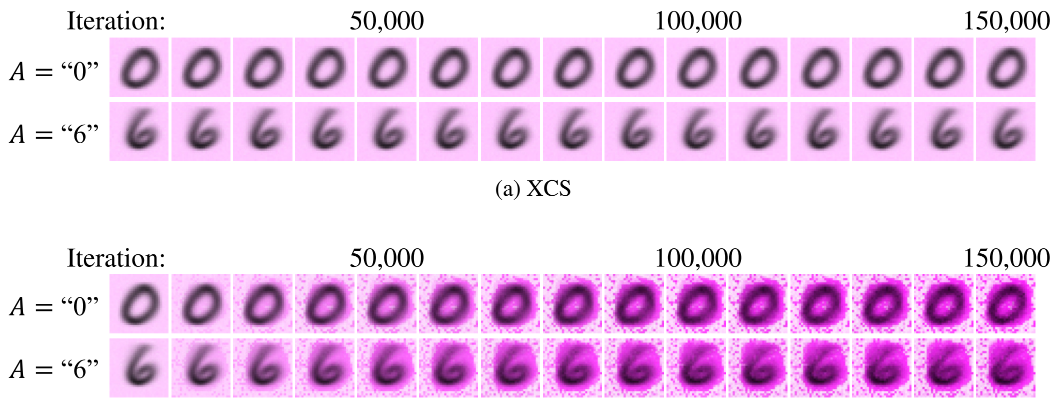


Fig. 9: Transition of the Average Condition of Classifiers in the Experiment 3 ($P_\# = 0.2$)

Fig. 10 shows the reward average of XCS-LCPCI with and without prediction inheritance in the 11-bit CIMUX problem. Note that XCS-LCPCI w/o Prediction Inheritance shows the same results as $P_\# = 0.6$ in Fig. 6. This result suggests that prediction inheritance has a negative impact on performance. This is caused by the influence of newly generated classifiers on the value of the prediction array. In the standard XCS-LCPCI, newly generated classifiers are excluded to determine the system output by the greedy selection because the prediction value of these classifiers are set to a value close to zero. In XCS-LCPCI with prediction inheritance, however, the covering classifier could copy classifiers having a high prediction value and generalize its condition without resetting the prediction. As a result, even transient classifiers are taken into account in the prediction array, which produces a noise in action selection. Consequently, XCS-LCPCI should not inherit the prediction value from the base classifier in classification problems.

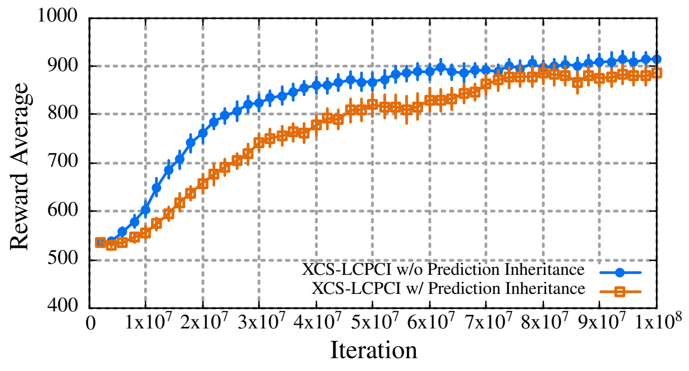


Fig. 10: Reward Average of XCS-LCPCI with and without Prediction Inheritance in the 11-bit CIMUX Problem

VII. Conclusion

In this paper, we proposed a niche-based covering method named Local Covering, which generates a new classifier by generalizing the nearest condition in the population. Also, to integrate Local Covering with XCS, we proposed XCS-LCPCI, which employs Local Covering only for the second or later covering for a certain state. The results of the three different experiments confirmed that XCS-LCPCI reduced the parameter sensitivity of $P_\#$.

Future work will include applying Local Covering to UCS [11], which forms a best action map instead of a complete action map. Local Covering could perform better with UCS because the comparison to a classifier predicting a zero reward never happens in a best action map. Also, the condition to be chosen as a base classifier of Local Covering needs to be discussed further. Furthermore, this method also can be applied to the real-valued XCS (XCSR) [12] by extending the concept of Minimum Hamming Distance (MHD), and this method has a potential to reduce the sensitivity of the parameter s_0 in XCSR.

References

[1] J. Holland, "Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems," in *Machine learning: An artificial intelligence approach*, R. Michalski, J. Carbonell, and T. Mitchell, Eds. Los Altos, CA: Morgan Kaufmann, 1986, vol. 2, ch. 20, pp. 593–623.

[2] S. W. Wilson, "Classifier fitness based on accuracy." *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[3] F. Kharbat, M. Odeh, and L. Bull, *Knowledge Discovery from Medical Data: An Empirical Study with XCS*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 93–121.

[4] N. Fredivianus, H. Prothmann, and H. Schmeck, "XCS revisited: A novel discovery component for the extended classifier system," in *Simulated Evolution and Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 289–298.

[5] M. V. Butz and S. W. Wilson, "An algorithmic description of XCS," in *Advances in learning classifier systems: Third international workshop, IWLCS 2000*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin Heidelberg: Springer-Verlag, 2001, pp. 253–272.

[6] A. Orriols-Puig and E. Bernadó-Mansilla, “Bounding XCS’s parameters for unbalanced datasets,” in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’06. New York, NY, USA: Association for Computing Machinery, 2006, p. 15611568.

[7] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” http://yann.lecun.com/exdb/mnist/, 2010.

[8] M. Nakata, W. Browne, T. Hamagami, and K. Takadama, “Theoretical XCS parameter settings of learning accurate classifiers,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO ’17, New York, NY, USA, 2017, pp. 473–480.

[9] M. V. Butz, “XCSJava 1.0: An implementation of the XCS classifier system in Java,” Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IlliGAL report 2000027, 2000.

[10] M. Tadokoro, S. Hasegawa, T. Tatsumi, H. Sato, and K. Takadama, “Knowledge extraction from XCSR based on dimensionality reduction and deep generative models,” in *2019 IEEE Congress on Evolutionary Computation (CEC)*, June 2019, pp. 1883–1890.

[11] E. Bernadó-Mansilla and J. M. Garrell-Guiu, “Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks,” *Evolutionary Computation*, vol. 11, no. 3, pp. 209–238, 2003.

[12] S. W. Wilson, “Get real! XCS with continuous-valued inputs.” in *Learning Classifier Systems*, ser. Lecture Notes in Computer Science, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., vol. 1813, 1999, pp. 209–222.